1/5/2 (Item 2 from file: 351) Links
Fulltext available through: Order File History
Derwent WPI

0010956888 & & *Drawing available*
WPI Acc no: 2001-580012/200165
XRPX Acc No: N2001-431819
Hardware description verifying system for software development, detects variation in logic interpretation of program portion, by discrepancies of compiled program from behavioral synthesis of hardware description
Patent Assignee: FURUSAWA S (FURU-I); NEC CORP (NIDE)
Inventor: FURUSAWA S

Patent Family (2 patents, 2 & countries)

Patent Number	Kind	Date	Application Number	Kind	Date	Update	Type
US 20010016935	A1	20010823	US 2001777543	A	20010206	200165	B
JP 2001222565	A	20010817	JP 200032355	A	20000209	200165	E

Priority Applications (no., kind, date): JP 200032355 A 20000209

Patent Details

Patent Number	Kind	Lan	Pgs	Draw	Filing Notes	
US 20010016935	A1	EN	27	19		
JP 2001222565	A	JA	12			

Alerting Abstract US A1
NOVELTY - A storage unit (4) stores a source program for hardware description in a programming language. A processor detects a portion of source program which is different in logic interpretation, by verifying discrepancies between a compiled program and behavioral synthesis of hardware description, and notifies the detected result to output unit (7).
DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

A. Hardware description verifying method;
B. Hardware description verifying program

USE - For determining discrepancies in logic interpretations between hardware description of programming languages such as C, C++, **Java** and the hardware description language (HDL), for software development.
ADVANTAGE - Enables detection of different portions of hardware description without examining the equality between functional verification of hardware descriptions and the behavior synthesis. Thus, the designer or user can easily rewrite the detected portion in the source program without any discrepancies.
DESCRIPTION OF DRAWINGS - The figure shows the block diagram of hardware description verifying system.
4 Storage unit
7 Output unit

Title Terms /Index Terms/Additional Words: HARDWARE; DESCRIBE; VERIFICATION; SYSTEM; SOFTWARE; DEVELOP; DETECT; VARIATION; LOGIC; INTERPRETATION; PROGRAM; PORTION; DISCREPANCY; COMPILE; SYNTHESIS

Class Codes

International Patent Classification

IPC	Class Level	Scope	Position	Status	Version Date

G06F-017/50			Main		"Version 7"
G06F-0017/50	A	I		R	20060101
G06F-0017/50	C	I		R	20060101

US Classification, Issued: 7164, 7165, 7166

File Segment: EPI;
DWPI Class: T01
Manual Codes (EPI/S-X): T01-F05A; T01-J15; T01-J15B; T01-S01C; T01-S02

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2001－222565
（P2001－222565A）
(43)公開日　平成13年８月17日(2001.8.17)

(51)Int.Cl.⁷	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　17/50		Ｇ０６Ｆ　15/60	６６４Ｚ　５Ｂ０４６ ６６４Ｇ

審査請求　有　　請求項の数16　ＯＬ　（全 12 頁）

(21)出願番号　特願2000－32355(P2000－32355)

(22)出願日　平成12年２月９日(2000.2.9)

(71)出願人　000004237
日本電気株式会社
東京都港区芝五丁目７番１号

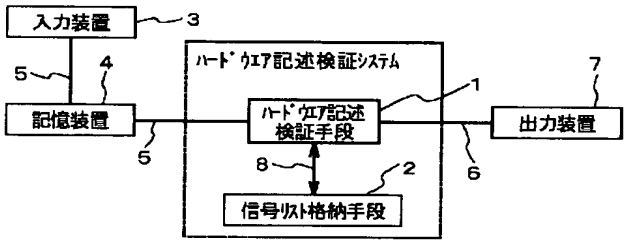
(72)発明者　古澤　慎也
東京都港区芝五丁目７番１号　日本電気株式会社内

(74)代理人　100102864
弁理士　工藤　実　（外１名）

Ｆターム(参考)　5B046 AA08 JA03 JA05 KA01

(54)【発明の名称】　ハードウエア記述の検証システム及びその検証方法

(57)【要約】
【課題】ハードウエア記述のうちプログラム言語とＨＤＬとで解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを回避する。
【解決手段】クロック文により値が更新されるプログラム言語で記述されるハードウエア記述をコンパイルするプログラム言語と、ハードウエア記述を動作合成するＨＤＬ言語との間で論理解釈が異なる部分を検出する。解釈が異なる部分は、クロックに同期して更新が起こる被代入変数が先行する他の数式を参照する部分、クロックに同期して更新が起こる被代入変数が重ね書きされる部分、同一クロック境界内でのみ有効であり同一クロック境界を越えるとその値が無効になる非代入変数が存在する部分、非オーバーライト型であり代入された値がその同一クロック境界内の全ての参照に反映される被代入変数が存在する部分等である。解釈が異なる部分を事前に発見することにより、検証の無駄が省略される。



1

【特許請求の範囲】

【請求項１】プログラム言語により記述したハードウエア記述をコンパイルする場合と、ＨＤＬ言語により記述したハードウエア記述を動作合成する場合とで、論理解釈が異なる部分を検出することを有するハードウエア記述の検証方法。

【請求項２】前記解釈が異なる部分は、被代入変数への代入後に前記被代入変数を他の数式で参照することが同一のクロックタイミング内で行われる部分である請求項１記載のハードウエア記述の検証方法。

【請求項３】前記プログラム言語により記述したハードウエア記述の複数文の一文を順番に入力するステップと、

前記一文が前記被代入変数への代入を行う文であると前記被代入変数を信号リストへ登録するステップと、

前記一文がクロック境界であれば登録されている前記被代入変数を前記信号リストから削除するステップと、

前記一文が前記信号リストに登録されている前記被代入変数を参照しているかどうかを判断するステップとを有する請求項２記載のハードウエア記述の検証方法。

【請求項４】前記一文が前記信号リストに登録されている前記被代入変数を参照している場合に、前記プログラム言語によるハードウエア記述と前記ＨＤＬ言語によるハードウエア記述との解釈が異なる動作となる警告を行うステップとを有する請求項３記載のハードウエア記述の検証方法。

【請求項５】前記解釈が異なる部分は、同一クロックタイミング内において被代入変数が非オーバーライト型として重ね書きされる部分である請求項１記載のハードウエア記述の検証方法。

【請求項６】前記プログラム言語により記述したハードウエア記述の複数文の一文を順番に入力するステップと、

前記一文が前記被代入変数への代入を行う文であると前記被代入変数を信号リストへ登録するステップと、

前記一文がクロック境界であれば登録されている前記被代入数変数を前記信号リストから削除するステップと、

前記一文が前記信号リストに登録されている前記被代入変数への代入を行っているかどうかを判断するステップとを有する請求項５記載のハードウエア記述の検証方法。

【請求項７】前記一文が前記信号リストに登録されている前記被代入変数への代入を行っている場合に前記プログラム言語によるハードウエア記述と前記ＨＤＬ言語によるハードウエア記述との解釈が異なる動作となる警告を行うステップとを有する請求項６記載のハードウエア記述の検証方法。

【請求項８】前記解釈が異なる部分は、同一クロックタイミング内でのみ値が有効であり、前記クロックタイミングを越えるとその値が無効になる非代入変数が存在する部分である請求項１記載のハードウエア記述の検証方法。

2

【請求項９】前記解釈が異なる部分は、非オーバーライト型である前記被代入変数へ代入された値が同一クロックタイミング内の全ての前記被代入変数の参照に反映される被代入変数が存在し、且つ、代入よりも参照が先になされている部分である請求項１記載のハードウエア記述の検証方法。

【請求項１０】前記解釈が異なる部分は、第１オペランドの条件判断が第２オペランドの評価に依存する、両オペランドを用いる演算子が存在する部分である請求項１記載のハードウエア記述の検証方法。

【請求項１１】プログラム言語により記述したハードウエア記述の文が一文ずつ入力されるハードウエア記述検証装置と、

信号リスト格納装置とを含み、

前記ハード記述検証装置は、前記プログラム言語によるハードウエア記述をコンパイルする場合とＨＤＬ言語によるハードウエア記述を動作合成する場合との間で動作の解釈が異なる部分を検出し、

前記信号リスト格納装置は、前記解釈が異なる部分を格納するハードウエア記述の検証システム。

【請求項１２】前記解釈が異なる部分は、被代入変数への代入後に前記被代入変数を他の数式で参照することが同一クロックタイミング内で行われる部分である請求項１１記載のハードウエア記述の検証システム。

【請求項１３】前記解釈が異なる部分は、同一クロックタイミング内において被代入変数が非オーバーライト型として重ね書きされる部分である請求項１１記載のハードウエア記述の検証システム。

【請求項１４】前記解釈が異なる部分は、同一クロックタイミング内でのみ値が有効であり、前記クロックタイミングを越えるとその値が無効になる非代入変数が存在する部分である請求項１１記載のハードウエア記述の検証システム。

【請求項１５】前記解釈が異なる部分は、非オーバーライト型である前記被代入変数へ代入された値が同一クロックタイミング内の全ての前記被代入変数の参照に反映される被代入変数が存在し、且つ、代入よりも参照が先にされている部分である請求項１１記載のハードウエア記述の検証システム。

【請求項１６】前記解釈が異なる部分は、第１オペランドの条件判断が第２オペランドの評価に依存する、両オペランドを用いる演算子が存在する部分である請求項１１記載のハードウエア記述の検証システム。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、ハードウエア記述の検証システム及びその検証方法に関し、特に、プログラム言語でハードウエアが記述されているハードウエア

記述について、ソフトウエアプログラムとしてコンパイルした場合と、ハードウエア動作記述として合成した場合とで、実質的に相違する恐れがある相違部分を発見するハードウエア記述の検証システム及びその検証方法に関する。

【０００２】

【従来の技術】従来、ハードウエアの設計はハードウエア記述言語（ＨＤＬ）を用いて行われ、そのハードウエアを制御するソフトウエアはプログラム言語を用いて記述されていた。ハードウエアとそのハードウエアを制御するソフトウエアプログラムとの統合検証は、協調シミュレータといわれるハードウエア記述言語とプログラム言語との混在実行が可能なシミュレータを用いて行われていた。

【０００３】このようなことは、ソフトウエアプログラムのデバッグ段階においても全く同様である。機能検証ステップＳ１０１において、ソフトウエアをハード上でシミュレーション動作させることでハードウエアを機能検証してそのハードウエアをデバッグするときにハードウエアを表すＨＤＬ記述とソフトウエアを表すプログラム言語とを用いて協動させると、ハードウエアシミュレーションの動作の遅さに検証時間が依存してしまう。最近は、そのような依存がなくソフトウエアもハードウエアもプログラム言語のみで記述し、高速検証を可能とするために、ＨＤＬの代わりにソフトウエアの開発に用いられるＣ、Ｃ＋＋、Ｊａｖａのようなプログラム言語によりハードウエア動作を記述して、ソフトウエアの検証を行うようになってきた（例えば、特開平１０－１４９３８２号）。

【０００４】一般に、回路設計を行うために必要となる処理は、図１９に示されるように２種類があり、それらは機能検証Ｓ１０１と動作合成Ｓ１０２とである。ステップＳ１０１は、そのハードウエア記述（プログラム言語による）をコンパイルして生成した実行形式で機能検証を実行する。ステップＳ１０２は、そのハードウエア記述（ＨＤＬ言語による）を動作合成してＲＴＬ（レジスタトランスファレベル）記述を生成するものである。

【０００５】ところが、ここで新たな問題が現れた。ソフトウエア用のプログラム言語をそのままハードウエア設計に適用することには不都合がある。例えば、回路の同時並列動作は通常のプログラム言語で記述され得ないという不都合がある。このような不都合を解消してハードウエア設計用のプログラム言語が適正であるように、そのプログラム言語はその動作仕様が拡張され言語拡張されている（発明の実施の形態の項で後述される。）。そのような言語拡張によって、同じハードウエア記述１００の動作の解釈が言語拡張をしたプログラム言語で記述したハードウエア記述を用いたＳ１０１の処理とＨＤＬで記述したハードウエア記述を用いた動作合成ツールの処理との間で異なっている可能性がある。このような

可能性が含まれているかどうかを確認するために、ステップＳ１０１の結果とステップＳ１０２の結果との等価性を確認する等価性検証のステップＳ１０３が新たに必要である。このような等価性検証により、同じハードウエア記述１００の動作の解釈がステップＳ１０１のコンパイラとステップＳ１０２の動作合成ツールとの間で異なっていることが判明しても、どのようにその解釈の相違が生じたか、どの記述部分でその相違が生じたのかが明らかでなく、人手を用いた多大な時間の投入による解析がハードウエアのデバッグに要する。

【０００６】ハードウエア記述のうちプログラム言語とＨＤＬとで解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを回避することが望まれる。特には、そのような等価性検証の不要化が望まれる。

【０００７】

【発明が解決しようとする課題】本発明の課題は、ハードウエア記述のうちプログラム言語とＨＤＬとで解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを回避することができるハードウエア記述の検証システム及びその検証方法を提供することにある。本発明の他の課題は、元々の原因がプログラム言語とＨＤＬとの違いを認識できずにハードウエア記述を両言語で記述してしまうことに起因して解釈が分かれるために必要となる等価性検証を不要化することができるハードウエア記述の検証システム及びその検証方法を提供することにある。

【０００８】

【課題を解決するための手段】本発明によるハードウエア記述の検証方法は、プログラム言語により記述したハードウエア記述をコンパイルする場合と、ＨＤＬ言語により記述したハードウエア記述を動作合成する場合とで、論理解釈が異なる部分を検出することである。論理解釈が異なる部分は、ハードウエア記述をシミュレートするコンピュータ設計者によりハードウエア記述のシミュレーションに先だって速やかに論理解釈が正しく確定され得る。

【０００９】その解釈が異なる部分は、被代入変数への代入後にその被代入変数を他の数式で参照することが同一のクロックタイミング内で行われる部分である。この場合、プログラム言語により記述したハードウエア記述の複数文の一文を順番に入力し、その一文が被代入変数への代入を行う文であればその被代入変数を信号リストへ登録し、その一文がクロック境界であれば登録されているその被代入数変数をその信号リストから削除し、その一文が信号リストに登録されている被代入変数を参照しているかどうかを判断する。その一文が信号リストに登録されている被代入変数を参照している場合に、プログラム言語によるハードウエア記述とＨＤＬ言語によるハードウエア記述との解釈が異なる動作となる警告が行われることになる。

【００１０】そのような解釈が異なる部分は、同一クロックタイミング内において被代入変数が非オーバーライト型として重ね書きされる部分でもある。この場合、そのハードウエア記述の複数文の一文を順番に入力し、その一文が被代入変数への代入を行う文であれば、その被代入変数を信号リストに登録し、その一文がクロック境界であれば登録されているその被代入変数をその信号リストから削除し、その一文が信号リストに登録されている被代入変数への代入を行っているかどうかが判断されることになる。更に、その一文が信号リストに登録されている被代入変数への代入を行っている場合にプログラム言語によるハードウエア記述とＨＤＬ言語によるハードウエア記述との解釈が異なる動作となる旨の警告が行われる。

【００１１】その解釈が異なる部分は、同一クロックタイミング内でのみ値が有効であり、そのクロックタイミングを越えるとその値が無効になる非代入変数が存在する部分でもあり、非オーバーライト型である被代入変数へ代入された値が同一クロックタイミング内の全ての被代入変数の参照に反映される被代入変数が存在し、且つ、その代入よりもその参照が先にされている部分でもあり、第１オペランドの条件判断が第２オペランドの評価に依存する、両オペランドを用いる演算子が存在する部分でもある。

【００１２】本発明によるハードウエア記述の検証システムは、プログラム言語により記述したハードウエア記述の文が一文ずつ入力されるハードウエア記述検証装置と、信号リスト格納装置とを含み、そのハードウエア記述検証装置は、プログラム言語によるハードウエア記述をコンパイルする場合とＨＤＬ言語によるハードウエア記述を動作合成する場合との間で動作の解釈が異なる部分を検出し、信号リスト格納装置は、その解釈が異なる部分を格納する。その解釈が異なる部分は、既述の通り例示されている。

【００１３】

【発明の実施の形態】図１に一致対応して、拡張プログラム言語により記述したハードウエア記述を検証する検証システムの実施の形態は、ハードウエア記述検証手段・ユニットが信号リスト格納手段・ユニットとともに設けられている。そのハードウエア記述検証手段１は、図１に示されるように、信号リスト格納手段２に双方向に接続している。入力装置３は、記憶装置４に接続している。入力装置３は、プログラム言語により記述されたハードウエア記述５を記憶装置４に出力する。記憶装置４は、入力されたハードウエア記述５を格納して記憶する。

【００１４】ハードウエア記述検証手段１は、ハードウエア記述５を記憶装置４から受け取って、後述される部分を検証（検出）したときに、その部分の存在とその存在箇所を信号化した検出信号を警告信号６として出力装置７に出力する。出力装置７は、その警告信号６を表示する表示装置である。信号リスト格納手段２は、検証対象を信号化した検証対象信号８を格納し、検証対象信号８をハードウエア記述検証手段１に出力する。

【００１５】検証対象は、５つの型に分けられる。５つの型は、発明者により、レジスタ型、非オーバーライト型、非レジスタ型、配線型、演算子存在型であるとそれぞれに呼ばれる。

レジスタ型：

```
clock　()
x＝a＋b
clock　()
y＝c－d
```

【００１６】このような原プログラムに含まれるｘとｙは、ともにプログラム上は変数であり、ハードウエアのイメージでは信号であるａとｂとにより、被代入変数ｘが記述されている。被代入変数ｙは、ｃとｄとにより同様に記述されている。クロックに同期して更新が起こる変数として定義されている被代入変数ｘ，ｙは、図２と図３にそれぞれに示されるように、クロック１１とクロック１２とに基づいてレジスタ１３とレジスタ１４に入力され、異なるクロックタイミングでそれぞれに変化し、その値の更新が代入時ではなくその直後にあるクロック記述に同期して起こる。このようにクロックタイミングで変化する被代入変数のこと（ｘ，ｙ）をプログラム言語上、レジスタ型と拡張的に定義しておく。これによって、実回路上のラッチやレジスタが表現される。

【００１７】

レジスタ型検証対象：

```
clock ()；
x＝a＋b；
y＝x＋t；
clock ()；
```

この原プログラムに含まれる２つのクロック記述で挟まれる１つの同じクロックタイミング内で代入された被代入変数ｘがその代入後に他の数式で参照されている。このような形態で２度記述された被代入変数は、ＨＤＬ記述を用いる動作合成では同じ値ではないと解釈されるが、言語拡張されたプログラム言語を解釈するコンパイラでは誤って同じ値であると解釈される。このような形態で２度以上記述されているレジスタ型被代入変数は、信号リスト格納手段２に格納される。クロックに同期して更新が起こる被代入変数がこのように同一タイミング内で代入された後に他の数式で参照される部分が、レジスタ型検証対象である。

【００１８】図５は、レジスタ型検証対象の存在を検出する検出方法を示している。図６は、レジスタ型検証対象を例示している。レジスタ型検証対象である原プログラムの拡張された仕様によるハードウエア記述は、次の通りである。

```
ｉｎｔ　ｔ；
ｒｅｇ　ｘ，ｙ；
ｘ＝０；・・・第１文
ｃｌｏｃｋ（）；・・・第２文
ｘ＝１；・・・第３文
ｔ＝３；・・・第４文
ｙ＝ｘ＋ｔ；・・・第５文
ｃｌｏｃｋ（）；・・・第６文
```

ここで、”ｒｅｇ　ｘ，ｙ”と第２文と第６文は、拡張された仕様によるプログラム言語表現である。

【００１９】記憶装置４からハードウエア記述の１文ずつがハードウエア記述検証手段１に入力される（ステップＳ１０１）。最初の一文は、”ｘ＝０”である。その入力文である第１文が、クロック境界を表す記述であれば、信号リスト格納手段２に格納されている信号情報の全てが削除される。第１文は、クロック境界ではないので、ステップＳ１０２からステップＳ１０４に進む。ステップＳ１０４では、その入力文が信号リスト格納手段２に格納されている信号を参照しているかどうかが判断される。今、信号リストには何も登録されていないので、プロセスはステップＳ１０６に進む。

【００２０】信号リスト格納手段２に格納されている信号を参照している場合は、ソフトウエア実行時に動作が意図しないものになる可能性があるので、その旨を警告としてハードウエア記述検証手段１は出力装置７に出力して、プロセスはステップＳ１０６に進む。ステップＳ１０６では、レジスタ型信号（ｘ）への代入が存在しているかどうかが判断される。第１文にはレジスタ型信号への代入が存在しているので、プロセスはステップＳ１０７に進む。

【００２１】ステップＳ１０７では、レジスタ型信号への代入が存在している場合、その一文で代入されている全てのレジスタ型信号情報を信号リスト格納手段２に格納する。第１文の中のレジスタ信号であるｘは信号リスト格納手段２に格納され、プロセスはステップＳ１０１に戻る。

【００２２】次に、第２文”ｃｌｏｃｋ”が、ハードウエア記述検証手段１に入力される。第２文はクロック境界であるから、信号リスト格納手段２に格納されている信号は全てが削除され、ステップＳ１０３でｘは削除される。プロセスは、ステップＳ１０１に戻る。

【００２３】次に、第３文”ｘ＝１”がハードウエア記述検証手段１に入力される。第３文は、クロック境界ではないので、プロセスはステップＳ１０４に進む。今、信号リストには何も登録されていないので、プロセスはステップＳ１０６に進み、ステップＳ１０７で信号リストへそのｘを登録する。

【００２４】次に、第４文”ｔ＝３”がハードウエア記述検証手段１に入力される。第４文はクロック境界ではないので、プロセスはステップＳ１０４に進み、第４文

のｔはｘを参照していないので、プロセスはステップＳ１０４からステップＳ１０６に進み、ステップＳ１０６では、レジスタ型被代入変数への代入は存在しないので、そのｔは登録されず、プロセスはステップＳ１０１に戻る。

【００２５】次に、第５文”ｙ＝ｘ＋ｔ”がハードウエア記述検証手段１に入力される。第５文は、クロック境界ではないので、プロセスはステップＳ１０４に進む。ステップＳ１０４で、第５文のｙは信号リスト格納手段２に登録されている信号ｘ（第３文のｘ）を参照しているので、ステップＳ１０５で警告が出力され、ステップＳ１０６ではレジスタ型被代入変数信号ｙへの代入が存在しているので、第５文のｙはステップＳ１０７で信号リスト格納手段２に登録される。次に、第６文がハードウエア記述検証手段１に入力され、ｘとｙとは信号リスト格納手段２から削除される。

【００２６】クロック境界内文である第３文と第５文では、第５文の式の被代入変数ｙが第３文である他の式を参照している。このような場合、Ｃ言語等によるコンパイラとＨＤＬでは、図６の表に示されるように、解釈が互いに異なっている。コンピュータ言語として仮に正しくそのプログラムが記載されていたとしても、言語拡張されたＣ言語によるこの記述と元々のＨＤＬとでは、解釈のなされ方が異なってしまうことがある。そのような場合の１つとして、同一のクロックタイミングで代入が行われた被代入変数（同じ文字が用いられている）を他の数式の中で参照する場合がある。

【００２７】このような場合、図７に示されるように、第２文のクロックにより、ＨＤＬではｘには０が代入された状態で第５文を実行するＣ言語ではｘには１が代入されるので、第５文では、Ｃ言語ではｙは４であるが、ＨＤＬではｙは３である。このように物理的動作がクロックによりレジスタ等で実行されるクロック境界の前後の文章で、Ｃ言語とＨＤＬでは解釈が異なり異なった演算を実行する。このような異なった解釈による演算の実行は、他の式を参照する第５文で結果が異なってしまう。

【００２８】

非オーバーライト型検証対象：

```
ｃｌｏｃｋ（）；
ｚ＝ａ＋ｂ；
ｚ＝ｃ＋ｄ；
ｃｌｏｃｋ
```

【００２９】このような原プログラムは、ハードウエアのイメージでは信号であるａとｂとｃとｄとにより２つのｚ，ｚが記述されている。このｚは、言語拡張において同一タイミング内において、オーバーライトされることがない変数として定義される。これによって、実回路上マルチプレクサを表現する。１つの被代入変数ｚはａとｂとにより記述され、他の１つの被代入変数ｚはｃと

ｄとにより記述されている。クロックに同期して更新が起こる変数として定義されている２つの被代入変数ｚ，ｚは、コンパイラでは上式のｚが下の式のｚに代入されて上書きされるので、下のｚのみが有効である。動作合成では、図４に示されるように、（ａ＋ｂ）と（ｃ＋ｄ）とがＭＰＸ（マルチプレクサ）１５により選択されることであると解釈される。非オーバーライト型では、同一クロックタイミング内において同一の被代入変数への代入は２度以上は許されない。

【００３０】図８は、非オーバーライト型検証対象の存在を検出する検出方法を示している。図９は、非オーバーライト型検証対象を例示している。非オーバーライト型検証対象である原プログラムの拡張された仕様によるハードウエア記述は、次の通りである。

【００３１】

```
ｔｅｒ　ｚ，ｔ；
ｚ＝０；・・・第１文
ｃｌｏｃｋ（）；・・・第２文
ｚ＝１；・・・第３文
ｔ＝３；・・・第４文
ｚ＝ｔ＋２；・・・第５文
ｃｌｏｃｋ（）；・・・第６文
```

ここで、”ｔｅｒ　ｚ，ｔ”と第２文と第６文は、拡張された仕様によるプログラム言語表現である。

【００３２】記憶装置４からハードウエア記述の１文ずつがハードウエア記述検証手段１に入力される（ステップＳ２０１）。最初の一文は、”ｚ＝０”である。その入力文である第１文が、クロック境界を表す記述であれば、信号リスト格納手段２に格納されている信号情報の全てが削除される（ステップＳ２０３）。第１文は、クロック境界ではないので、ステップＳ２０２からステップＳ２０４に進む。

【００３３】ステップＳ２０４では、その入力文が信号リスト格納手段２に格納されている信号への代入が存在しているかどうかが判断される。今、信号リストには何も登録されていないので、プロセスはステップＳ２０６に進む。その代入が存在している場合には、警告が発せられた後にステップＳ２０６に進む。

【００３４】ステップＳ２０６では、非オーバーライト型信号ｚへの代入が存在しているかどうかが判断される。第１文には非オーバーライト型信号ｚへの代入が存在しているので、プロセスはステップＳ２０７に進む。ステップＳ２０７では、非オーバーライト型信号への代入が存在している場合、その一文で代入されている全ての非オーバーライト型信号を信号リスト格納手段２に格納する。第１文の中の非オーバーライト型信号であるｚは信号リスト格納手段２に格納され、プロセスはステップＳ２０１に戻る。

【００３５】次に、第２文”ｃｌｏｃｋ”が、ハードウエア記述検証手段１に入力される。第２文はクロック境界であるから、信号リスト格納手段２に格納されている信号は全てが削除され、ステップＳ２０３でｚは削除される。プロセスは、ステップＳ２０１に戻る。

【００３６】次に、第３文”ｚ＝１”がハードウエア記述検証手段１に入力される。第３文は、クロック境界ではないので、プロセスはステップＳ２０４に進む。ステップＳ２０４では、今信号リストには何も登録されていないので、ステップＳ２０６に進み、ステップＳ２０７でｚを信号リストへ登録する。

【００３７】次に、第４文”ｔ＝３”がハードウエア記述検証手段１に入力される。第４文はクロック境界ではないので、プロセスはステップＳ２０４に進む。ステップＳ２０４では、信号リストにはｚのみが登録されているがｔは登録されていないので、プロセスはステップＳ２０６に進み、ステップＳ２０６では、ｔはオーバーライト型信号への代入が存在しているので、そのｔは登録され、プロセスはステップＳ２０１に戻る。

【００３８】次に、第５文”ｚ＝ｔ＋２”がハードウエア記述検証手段１に入力される。第５文は、クロック境界ではないので、プロセスはステップＳ２０４に進む。ステップＳ２０４で、第５文のｚは信号リスト格納手段２に登録されている被代入信号であるので、ステップＳ２０５で警告が出力され、ステップＳ２０６で非オーバーライト型信号ｚへの代入が存在しているので、第５文のｚはステップＳ２０７で信号リスト格納手段２に登録される。次に、第６文がハードウエア記述検証手段１に入力され、ｔとｚとは信号リスト格納手段２から削除される。

【００３９】クロック境界内文である第３文と第５文では、第５文の式の被代入変数ｚが第３文のｚでオーバーライトされる可能性がある。このような場合、Ｃ言語によるコンパイラとＨＤＬでは、図９の表に示されるように、解釈が互いに異なっている。このような場合、第２文のクロックにより、ＨＤＬではｚに０が代入されるが、Ｃ言語では第３文ではｚは１、第５文ではｚは５であるが、図１０に示されるように、ＨＤＬでは第３文と第５文とのどちらが有効になるか分からないので、このクロック内ではｚは不明である。このように物理的動作がクロックによりレジスタで実行されるクロック境界の前後の文章で、Ｃ言語とＨＤＬでは解釈が異なり異なった演算を実行し、又は、その演算は実行され得ない。このような異なった解釈による演算の実行・不実行は、オーバーライトすると解釈するか、不明として解釈するかで結果が異なってしまう。

【００４０】このように、レジスタ型では、信号代入が行われてもすぐには値の更新が行われずクロック境界記述により一斉に値の更新が行われ、非オーバーライト型では、連続したクロック境界記述に挟まれた部分では同一信号に対して複数回の代入を許されない。クロック文章の導入による拡張仕様のプログラム言語では、これに

物理的記述が入り込んでいて、設計者の意図は物理的記述に反映されていない。ハードウエア記述の機能検証と、ハードウエア記述の動作合成の機能検証との等価性の確認をしないで、その等価性が崩れる恐れがある記述部分を検出することにより、原プログラムのその恐れある部分に関して、設計者又はユーザーが解釈に分かれが生じないように書き換える。そのように書き換えられた原プログラムには、解釈の相違は存在しなくなるので、等価性の確認検証は省略され得る。

【００４１】以後の型でも以上に既述した同様の各定義を行い、定義にそぐわない各問題を警告する。図１１は、非レジスタ型検証対象を例示している。非レジスタ型検証対象である原プログラムの拡張された仕様によるハードウエア記述は、次の通りである。

非レジスタ型検証対象：

```
ｔ＝３；
ｃｌｏｃｋ（）；
ｚ＝ｔ＋２；
ｃｌｏｃｋ（）；
```

この原プログラムに含まれるｔは、２つのクロックで挟まれる１つの同じクロック境界内で記述されている。３が代入されたｔは、そのステップであるそのクロック境界内でのみ有効であり、クロック境界を越えるとその値の３が無効になるタイプの信号であると拡張定義し、非レジスタ型と呼ばれる。

【００４２】Ｃ言語では以前に代入された値が使用されるが、ＨＤＬでは同一ステップ内にｔへの代入がないので、図１２に示されるように、使用される値は合成ツールに依存する。このため、同一ステップ内で、且つ、参照以前に代入されていない非レジスタ型信号ｔを参照するｚの値は、Ｃ言語とＨＤＬでは異なってしまう。図１１の表に示されるように、Ｃ言語ではｚ＝５であるが、ＨＤＬではｚは不明である。第３文”ｚ＝ｔ＋２”は、信号リストに登録されていない非レジスタ型信号ｔを参照しているので、図１３のステップＳ３０１とステップＳ３０２に示されるように警告が発せられ、非レジスタ型信号への代入が存在しているので（ステップＳ３０３）、その非レジスタ型信号ｚは信号リストに登録される（ステップＳ３０４）。

【００４３】図１４は、配線型検証対象を例示している。配線型検証対象である原プログラムの拡張された仕様によるハードウエア記述は、次の通りである。

配線型検証対象：

```
ｔ＝３；
ｃｌｏｃｋ（）；
ｚ＝ｔ＋２；
ｔ＝１；
ｃｌｏｃｋ（）；
```

この原プログラムに含まれるｔは、２つのクロックで挟まれる１つの同じクロック境界内で２度記述されている。代入された値がその代入されたのと同一クロックタイミング内の全ての参照に反映されるタイプの信号ｔは、配線型と呼ばれる。実回路上、信号配線を表現する。

【００４４】Ｃ言語では以前に代入された値が使用されるが、ＨＤＬでは同一ステップ内でｔへ代入された値が使用される。図１５に示されるように、ｔに代入される値は、以前に代入された値の３であるか、新たに代入される値の１であるかが、Ｃ言語とＨＤＬとでは異なってしまう。図１４の表に示されるように、Ｃ言語ではｚ＝５，ｔ＝１であるが、ＨＤＬではｚ＝３，ｔ＝１である。第３文”ｚ＝ｔ＋２”は、信号リストに登録されていない配線型信号ｔを参照しているので、図１６のステップＳ４０１とステップＳ４０２に示されるように警告が発せられ、配線型信号への代入が存在しているので（ステップＳ４０３）、その配線型信号ｚは信号リストに登録される（ステップＳ４０４）。

【００４５】図１７は、演算子型検証対象を例示している。演算子型検証対象である原プログラムの拡張された仕様によるハードウエア記述は、次の通りである。

演算子型検証対象：

```
ｉ＝０；
ａ＝０；
ｃｌｏｃｋ（）；
ｉｆ（ｉ＞０&&ａ＋＋）｛
　　ｉ＝０；
｝
ｃｌｏｃｋ（）；
```

【００４６】まず、ａ＋＋とは、これを評価するとａをインクリメントすることを意味する。この原プログラムに含まれる演算子＆＆は、Ｃ言語では演算子＆＆の左オペランド（ｉ＞０）の条件が眞である場合にのみその右オペランドａ＋＋を評価するが、ＨＤＬでは演算子＆＆の左オペランドの条件の真偽に係わらず左右のオペランドを並列に評価する。演算子＆＆の左オペランドの条件が偽の場合に、右オペランドを評価するかどうかの点が、Ｃ言語とＨＤＬとでは異なる。図１８に示されるように、演算子＆＆が使用されていて（ステップＳ５０１）、演算子＆＆の右オペランド中に変数値の更新を伴う記述が存在する場合（ステップＳ５０２）、警告が出される（ステップＳ５０３）。演算子＆＆に代えられて左オペランドの条件が偽の場合にのみ右オペランドを評価する演算子が用いられている場合にも、図１８に示されるように警告が発せられる。

【００４７】

【発明の効果】本発明によるハードウエア記述の検証システム及びその検証方法は、ハードウエア記述のうちプログラム言語とＨＤＬとで解釈が分かれる部分を検出することにより、そのように解釈が分かれる部分の存在が機能検証シミュレーションを妨害することを警告するこ

13

とによってプログラム修正を容易化し、ひいては回避することができる。特には、等価性検証を省略することができる。

【図面の簡単な説明】

【図１】図１は、本発明によるハードウエア記述の検証システムの実施の形態を示すシステムブロック図である。

【図２】図２は、ハードウエア記述図である。

【図３】図３は、他のハードウエア記述図である。

【図４】図４は、更に他のハードウエア記述図である。

【図５】図５は、本発明によるハードウエア記述の検証方法の実施の形態を示すフロー図である。

【図６】図６は、両言語の記述の内容を示すデータ表である。

【図７】図７は、更に他のハードウエア記述図である。

【図８】図８は、本発明によるハードウエア記述の検証方法の実施の他の形態を示すフロー図である。

【図９】図９は、両言語の記述の他の内容を示すデータ表である。

【図１０】図１０は、更に他のハードウエア記述図である。

【図１１】図１１は、両言語の記述の内容を示すデータ表である。

【図１２】図１２は、更に他のハードウエア記述図である。

14

【図１３】図１３は、本発明によるハードウエア記述の検証方法の実施の更に他の形態を示すフロー図である。

【図１４】図１４は、両言語の記述の内容を示すデータ表である。

【図１５】図１５は、更に他のハードウエア記述図である。

【図１６】図１６は、本発明によるハードウエア記述の検証方法の実施の更に他の形態を示すフロー図である。

【図１７】図１７は、両言語の記述の内容を示すデータ表である。

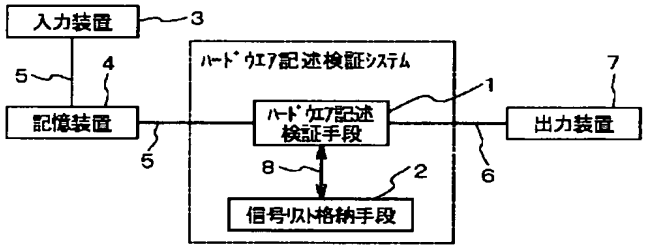
【図１８】図１８は、本発明によるハードウエア記述の検証方法の実施の更に他の形態を示すフロー図である。

【図１９】図１９は、公知のハードウエア記述の検証システムを示す回路システム図である。

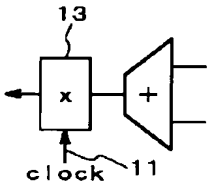
【符号の説明】

1…ハードウエア記述検証手段
2…信号リスト格納手段
3…入力装置
4…記憶装置
5…ハードウエア記述
6…警告信号
7…出力装置
8…検証対象信号

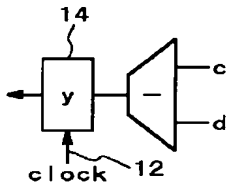
【図１】



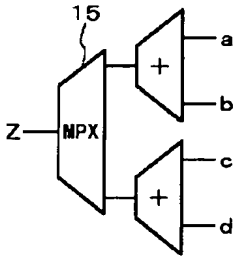
【図２】



【図３】



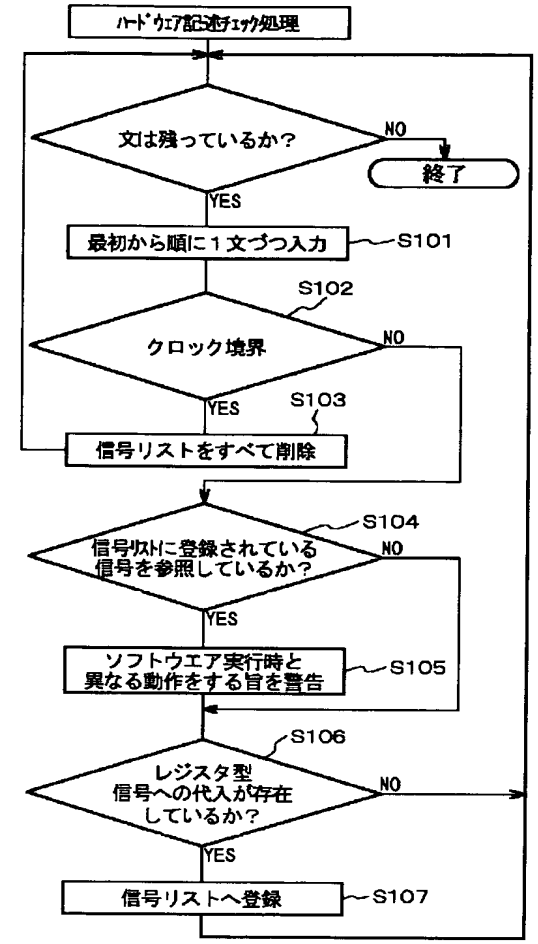
【図４】



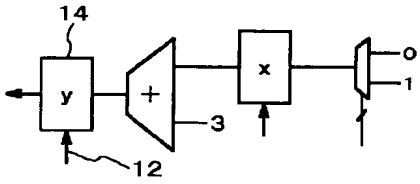
【図６】

```
/* reg.C */

#ifdef C
# define reg unsigned int
#endif

main()
{                                /* C */          /* HDL */        /* LIST */
        int t;
        reg x, y;

        x = 0;                   /* x=0 */                           /* {x} */        /* S107 */
        clock();                                  /* x=0 */          /* {} */         /* S103 */
        x = 1;                   /* x=1 */                           /* {x} */        /* S107 */
        t = 3;                   /* t=3 */          /* t=3 */        /* {x} */
        y = x + t;               /* y=4 */                           /* {x,y} */      /* S105,S107 */
        clock();                                  /* x=1,Y=3 */      /* {} */         /* S103 */
```

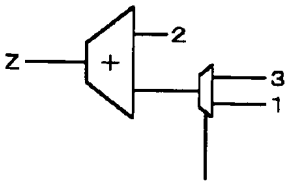
【図５】



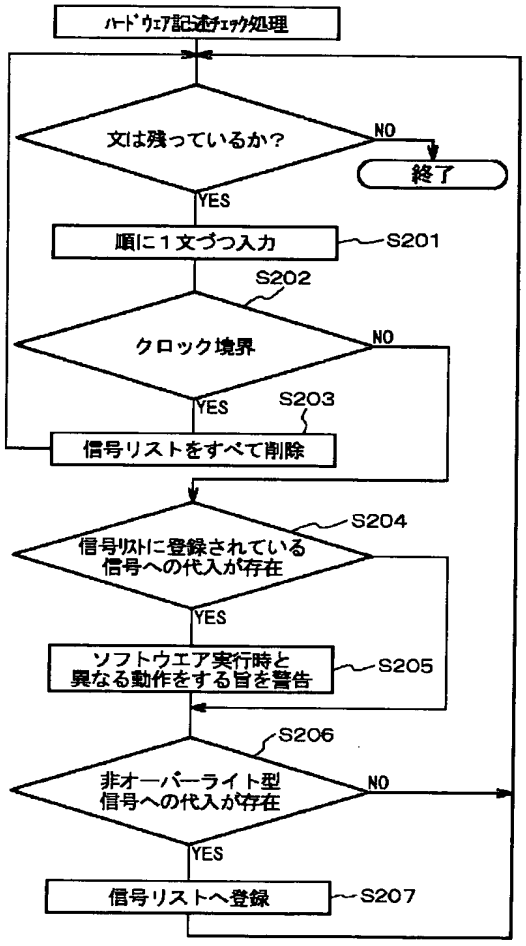
【図７】



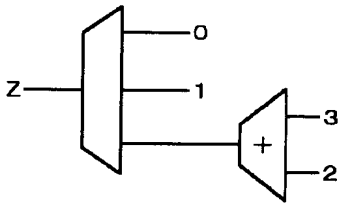
【図１５】



【図８】



【図１０】



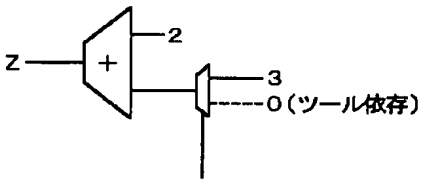
【図１１】

```
/* ter.C */

#ifdef C
# define ter unsigned int
#endif

main()
{                              /* C */          /* HDL */        /* LIST */
        ter z, t;

        t = 3;                 /* t=3 */        /* t=3 */        /* {t} */
        clock();                                /* t=? */        /* } */
        z = t + 2;             /* z=5 */        /* z=? */        /* {z} */
        clock();                                                /* } */
```

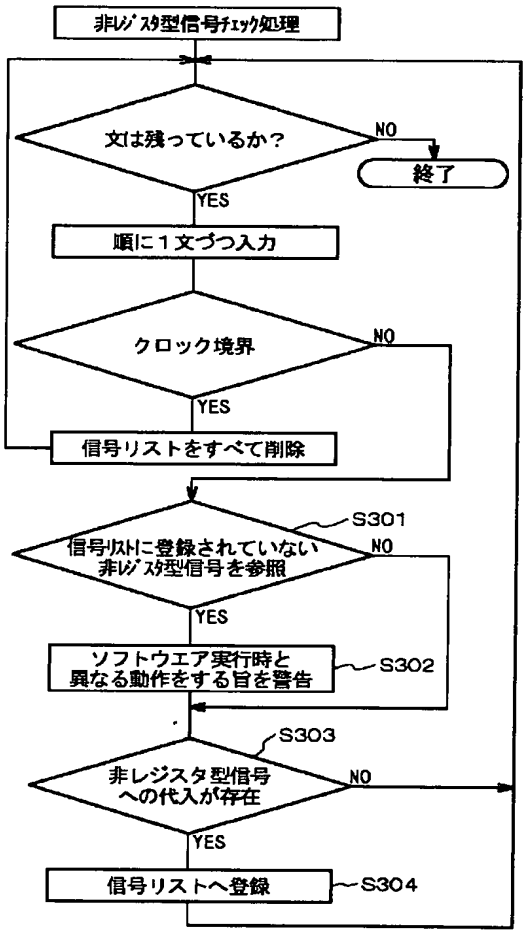
【図９】

```
/* assign.C */

#ifdef C
# define ter unsigned int
#endif

main()
{                       /* C */         /* HDL */       /* LIST */
        ter z, t;

        z = 0;          /* z=0 */       /* z=0 */       /* {z} */       /* S207 */
        clock();                                        /* {} */        /* S203 */
        z = 1;          /* z=1 */       /* z=? */       /* {z} */       /* S207 */
        t = 3;          /* t=3 */       /* t=3 */       /* {z,t} */     /* S207 */
        z = x + t;      /* z=4 */       /* z=? */       /* {z,t} */     /* S205,S207 */
        clock();                                        /* {} */        /* S203 */
}
```

【図１２】



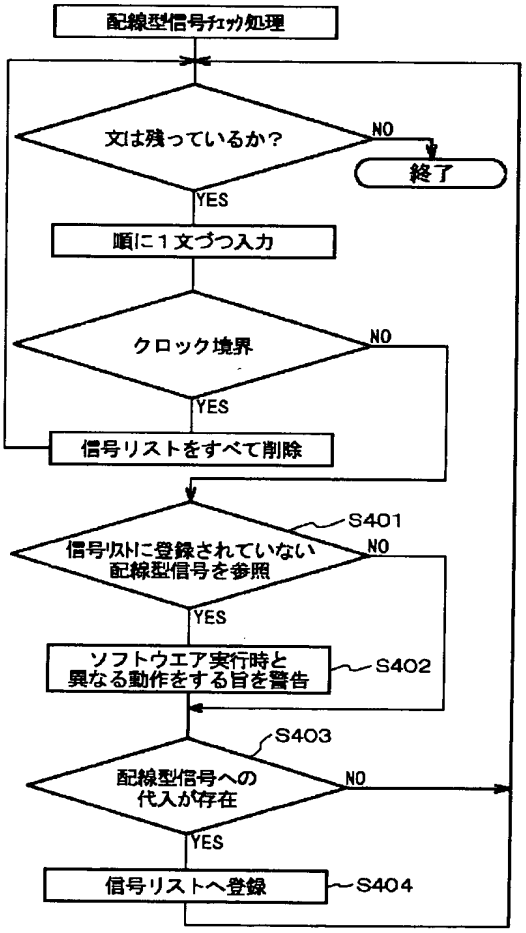
【図１３】



【図１４】

```
/* wire.C */

#ifdef C
# define ter unsigned int
#endif

main()
{                          /* C */          /* HDL */        /* LIST */
        ter z, t;

        t = 3;             /* t=3 */        /* t=3 */
        clock();                                             /* {t} */
        z = t + 2;         /* z=5 */        /* z=3 */        /* {} */
        t = 1;             /* t=1 */        /* t=1 */        /* {z} */
        clock();                                             /* {z,t} */
                                                             /* {} */
```

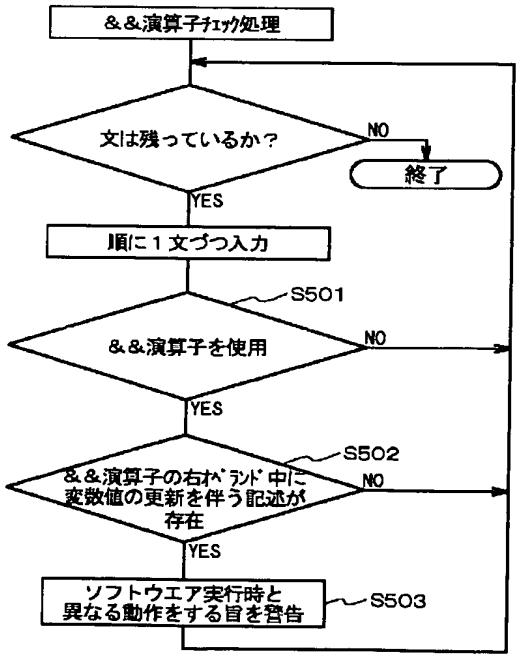
【図１６】



【図１７】

```
/* and.C  */

main()
{                          /* C */          /* HDL */
        int a,i;

        i = 0;             /* i=0 */        /* i=0 */
        a = 0;             /* a=0 */        /* a=0 */
        clock();
        if(i > 0 && a++) {     /* a=0 */        /* a=1 */
                i = 0;
        clock();
```

【図１８】



【図１９】